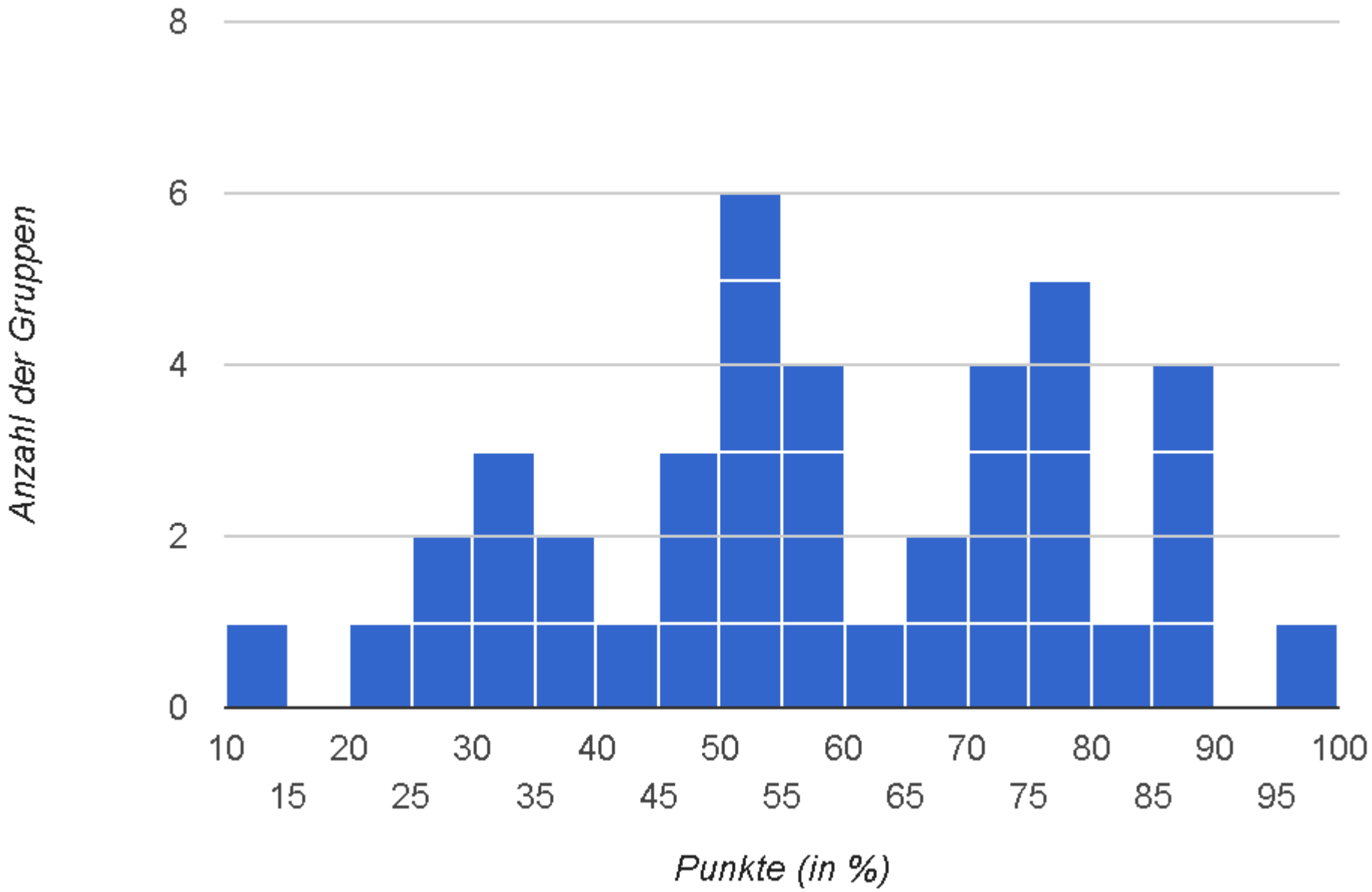
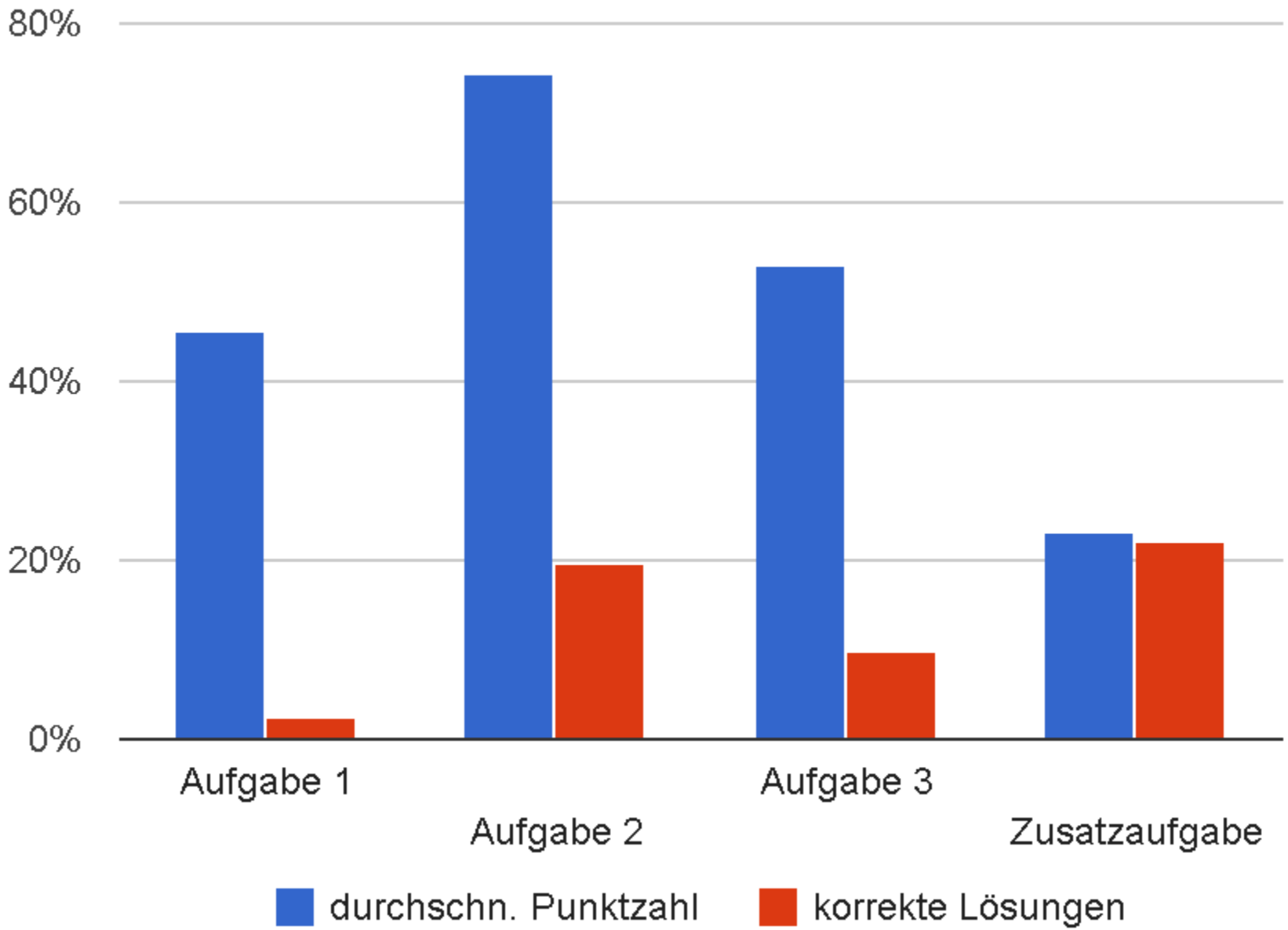
Übungsblatt 5 - Auswertung

Grundlegende Datenstrukturen

Punktverteilung (Gesamt)



Ergebnisse pro Aufgabe



Aufgabe 1 - Sequenzen

- Reine Fleißaufgabe
- Interfaces
  - Stack
  - Queue
  - List
- Klassen
  - ArraySequence
  - SinglyLinkedSequence
  - DoublyLinkedSequence
  - CircularBuffer (*extends ArraySequence*)

als Liste

```
public void push(T element) {
  insert(length(), element);
}

public T pop() {
  return remove(length()-1);
}
```

```
public void enqueue(T element) {
  insert(length(), element);
}

public T dequeue() {
  return remove(0);
}
```

Aufgabe 1 - Sequenzen

- Reine Fleißaufgabe
- Interfaces

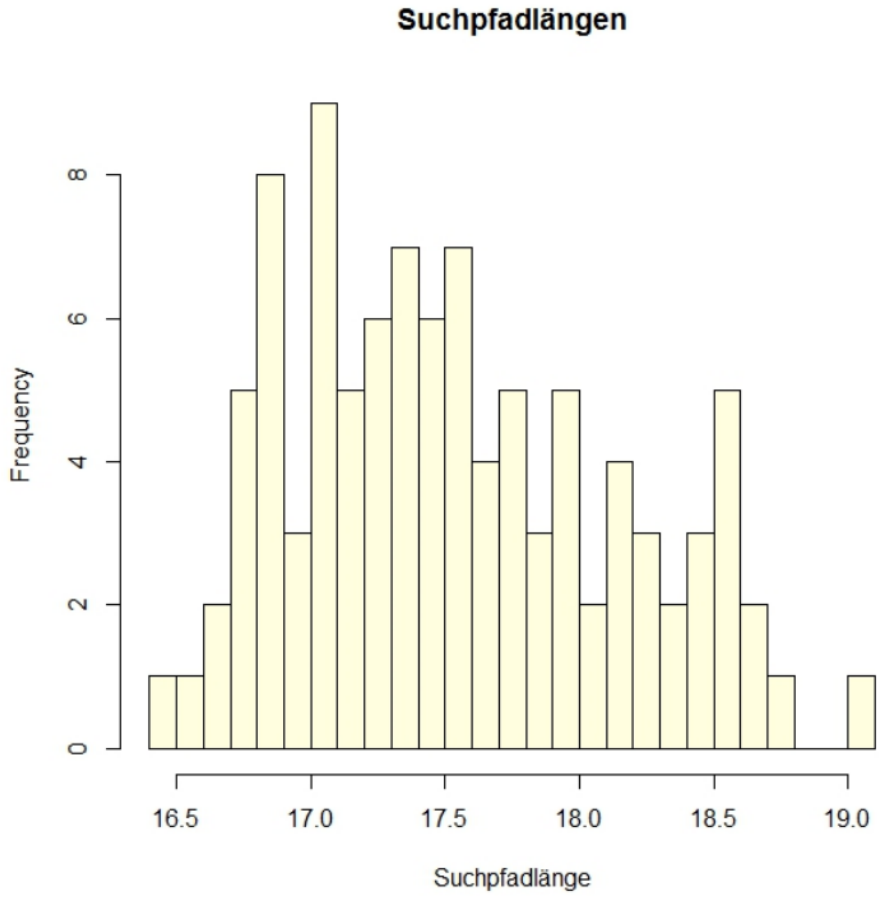
```java
public void push(T element) {
  insert(length(), element);
}

public T pop() {
  return remove(length()-1);
```

```java
public class ArraySequence<T> implements List<T>, Queue<T>, Stack<T> {
  T[] data;
  int size;
  // ...

  public T get(int index) {
    if(index<0 || index>=size) throw new IllegalStateException();
    return data[index];
  }

  public T remove(int index) {
    T old = get(index);
    if(index+1<size)
      // move data from the end of the array
      System.arraycopy(data, index+1, data, index, size-index-1);

    size--;
    data[ size] = null;
    return old;
  }
  // ...
```

```java
public class ArraySequence<T> implements List<T>, Queue<T>, Stack<T> {
  T[] data;
  int size;
  // ...

  public void insert(int index, T item) {
    // max index = current size
    if(index<0 || index>size) throw new IllegalStateException();

    T[] previous = data;
    if(size==data.length) {
      // increase array
      previous = data;
      data = createArray(2 * previous.length);

      if(index>0)
        // copy data at the beginning of the array
        System.arraycopy(previous, 0, data, 0, index);
    }

    // index not at the end of the array?
    if(index<size)
      // copy data at the end of the array
      System.arraycopy(previous, index, data, index+1, size-index);

    // set new value
    data[index] = item;
    size++;
  }
}
```

Aufgabe 2 - Bäume

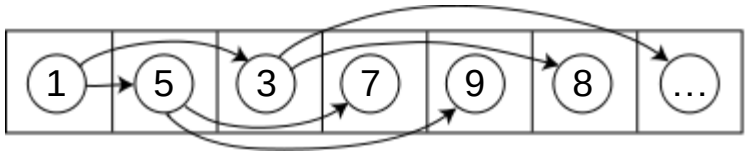
- Implementierung okay
- Was ist ein Histogramm?
  - X=10
  - 100 Durchläufe
  - Suchpfadlänge ~17.53

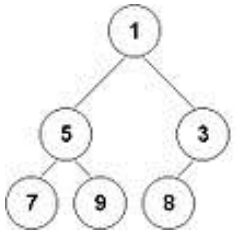


*D.Birnstiel und K.Utecht

Aufgabe 3 – Heaps

- MinHeap als Array vs. Baum





Kekule- oder auch
Sosa-Nummerierung

```java
public class MinHeapArray<T extends Comparable<T
  T[] array;
  int size;
  // ...
  private int parent(int i){
    return  (i - 1) / 2;
  }
  private int left(int i){
    return  2 * i + 1;
  }
  private int right(int i){
    return  2 * i + 2;
  }
  private int last(int i){
    return size;
  }
  // ...
}
```

```java
public class MinHeapTree<T extends Comparable<T>> implements MinHeap<T> {
  public class Node<T extends Comparable<T>> {
    private Node<T> left, right, parent;
    private T data;
  }
  Node<T> head;
  int size;
  // ...
  public Node<T> append(T value){
    // height of the tree - 1
    int h = (int) (Math.log(size()+1)/Math.log(2));
    // position in the last layer
    int p = (int)(size() - Math.pow(2, h) + 1);
    // traverse tree
    Node<T> current = head;
    for(;h>1;h--) {
      current = (p&((int)Math.pow(2, h-1)))==0?current.left:current.right;
    }
    Node<T> newNode;
    // append new element
    if((p&1)==0) {
      newNode = current.left = new Node<T>(current,value);
    } else {
      newNode = current.right = new Node<T>(current,value);
    }
    return newNode;
  }
}
```

Zusatzaufgabe

- Maulwurfsbau mit Luftzug
- Gegeben:
  - Ein Graph mit Knoten (*label ϵ {0,1}* → *Ausgang?*)
- Gesucht:
  1. Belüftete Knoten
  2. Belüftete Paare
- Graph besteht aus unabhängigen Teilgraphen
- Pro Teilgraph
  1. Starte von zufällig belüfteter Wurzel *
  2. Tiefentraversierung ⟶
  3. Zähle belüftete Knoten *K* auf dem Rückweg +

  Belüftete Paare: *(K*(K-1))/2)*